Configuration and Tooling

Tools

Build tools

The eagle-i ontology is built by Bamboo whenever there is a change to the files and externals described above. The build job can be found at: https://build.eagle-i.net/browse/DATAMODEL-JAR. This job builds the Jar file described above and sends it to the Nexus server to make it available to subsequent application builds. To generate the same files locally, the various scripts in the src/isf/module-scripts can be used to generate the same results as Bamboo. The Bamboo job uses these scripts through a Bamboo specific wrapper script that is executed during the Maven build. There is a set of scripts for Windows (the *.bat scripts) and the others are for Linux/Mac. The tools and scripts only need Java on the local system, you don't need Maven installed unless you would like to use the Maven build that Bamboo executes.

Issue tracker

Issues in the previous eagle-i Google Code site have been moved to the ISF ontology GitHub repository. Further issues will be managed there: https://github.com/vivo-isf/vivo-isf-ontology/issues.

Modules

A module configuration is a way for defining how to generate an output OWL file from other input OWL files. The configuration files are OWL files. One of the OWL files of a module is a "configuration" file for the tool to describe how the output OWL file should be generated from the input OWL files. The full details of a module configuration, the tools, and all available options is outside the scope of this document. The website for the tools will provide this information since it is an independent project that is used by other ontology projects and users.

> More tool documentation is available at: **https://github.com/ShahimEssaid/OwlCL**
> The pre-built tools we are using with eagle-i are from the eagle-i branch here: **https://github.com/openrif/tools/tree/eaglei**

Overview of Common Editing tasks

Application configuration

Application specific configuration is done by editing the application files as needed and then committing to SVN. The steps would be something like this:

1. Checkout or update SVN.
2. If the changes require some ontology content, for example:
3. if a new technique was added to the ISF by someone else, and
4. eagle-i would like to override the ISF label with a custom preferred application label
5. then before the application label can be added to the new class, the new class has to be visible when opening the application file.
6. For this to happen, the eagle-i module(s) will have to be generated or regenerated using the module scripts so that the generated module files will contain the newly added class. Alternatively, in the application file, a new class can be created with same IRI as in the ISF and then application specific annotations can be added on this new class. This would avoid the need for a local generation to see this class. However, doing this with the generation scripts will guarantee that things are working as expected and to make sure that the Bamboo build will work as expected.
7. At this point, the new preferred label can be added, or other application specific changes can be made.
8. Review changes in your SVN tool and then commit.
9. This will trigger a Bamboo build.

> This workflow will:
>
> 1. Not affect the ISF files
> 2. Does not require local module builds unless the latest modules are needed to be able to apply the application configuration changes.
> 3. The details for how to configure the application are based on specific application annotations, where they are made in the application files, etc. See other documentation sections for details.

Ontology configuration

To change the "eagle-i ontology" you will need to change the appropriate "module configuration" in order to affect the generated ontology modules. Or change the ISF content that is being used by the current "module configurations".

Changing the ISF

For example, if you need to add a new technique term, this change needs to be done in the ISF unless it is considered to be a technique that is too specific to eagle-i and it should not be made in the ISF. The top class "technique" is already configured to be included in the eagle-i build (with all its subclasses) so the only change needed is to add it to the ISF. To do this:

a. Checkout or update the eagle-i SVN.
b. This will also checkout in the latest ISF in the ISF folder described above.

        i. If eagle-i pins the ISF external to a specific version, it will not be possible to do the steps described here.
    c. Open the isf-dev.owl file from the ISF folder and then change to the appropriate ISF file and add the term. (To determine the appropriate ISF file, see which file is currently used for the parent class and use the same file.)
    d. Review changes (in Protege and / or in your SVN tool) to make sure they are correct.
    e. Now you need to commit the change to the ISF
        i. Most SVN tools will require that the commit is done from inside the external folder.
        ii. If you are using a command line tool, change to the root ISF folder and do a “commit”
        iii. If you are using a GUI, right click on the root ISF folder and do the commit.
    f. The Bamboo build should be triggered and the new technique should be included in the generated files.

Changing module configuration

The other type of change involves changing the module configuration without changing the ISF. For example, there might be a new hierarchy in the ISF that wasn’t used in eagle-i but should be used now. The steps would be:

a. Decide which eagle-i module configuration should be adjusted. For example, if the ero.owl file is the one that needs the new hierarchy, the “eagle-i” module configuration needs to be changed.
b. Open the configuration file for the specified module and make any necessary configuration changes. The details are in the tools documentation.
c. Before committing to SVN, generate the ontology files locally to verify that the configuration changes will give you the desired results.
d. When the configuration is correct, commit the changes to SVN. This will trigger the Bamboo build and the result should match the results of the generated ontology.